



Master Big Data & Internet of Things

Academic Year 2025-2026

Arabic Sentiment Analysis

Comparative Study of Machine Learning and Deep Learning Methods

Prepared by:

Amine Hamouchi
Ghoufrane Bendahmane

Supervised by:

Said Bahassine

May 2026

Abstract

This report presents a comparative study of Arabic sentiment classification. The work covers dataset preparation, Arabic text preprocessing, text representation, feature selection, model training, and critical analysis of experimental performance across machine learning and deep learning approaches.

Keywords: NLP, Arabic, sentiment analysis, classification, machine learning, deep learning.

Contents

1	General Introduction	1
2	Related Work	2
3	Dataset Description	3
3.1	Arabic tweets corpus (data/Tweets.txt)	3
3.1.1	Origin and format	3
3.1.2	Labels and interpretation	3
3.1.3	Class distribution and implications	4
3.1.4	Typical content (qualitative)	4
3.2	Balanced hotel reviews (data/balanced-reviews.txt)	4
3.2.1	Origin and format	4
3.2.2	Star ratings	5
3.2.3	Linguistic characteristics	5
3.3	Summary comparison	5
4	Data Preprocessing	6
4.1	Pipeline for tweets (preprocess_tweet)	6
4.1.1	Usage (code)	6
4.2	Pipeline for hotel reviews (preprocess_balanced_review)	7
4.3	Binary labels for star ratings	7
4.4	Illustrative text transformations	7
4.5	Empirical effect of cleaning (full-corpus statistics)	8
4.6	Discussion: differences between corpora after cleaning	8
4.7	Optional stemming	8
5	Text Representation	9
5.1	Problem setting	9
5.2	Bag of Words (BoW)	9
5.2.1	Principle	9
5.2.2	n -grams and vocabulary control	10
5.2.3	Strengths and limits	10
5.3	Term Frequency-Inverse Document Frequency (TF-IDF)	10
5.3.1	Motivation	10
5.3.2	Components	10
5.3.3	Normalisation	10
5.3.4	Relation to BoW	10
5.4	Distributed representations: FastText and AraVec	11
5.4.1	Word embeddings	11
5.4.2	Subwords and Arabic	11

5.4.3	Document vectors	11
5.5	Contextual representations: AraBERT	11
5.5.1	From words to transformers	11
5.5.2	Fine-tuning	11
5.5.3	Arabic-specific benefit	11
5.6	Choosing a representation for this project	11
6	Feature Selection	13
6.1	Notation	13
6.2	Methods implemented	13
6.2.1	Baseline: no selection (<code>none</code>)	13
6.2.2	Chi-square test (<code>chi2</code>)	13
6.2.3	Mutual information (<code>mi</code>)	14
6.2.4	Fisher score ranking (<code>fisher</code>)	14
6.3	Selecting the best method: cross-validation over a discrete grid	14
6.4	Code: running the comparison	15
6.5	Results	16
6.6	Choosing k (number of retained features)	16
7	Models	17
7.1	Machine Learning Models	17
7.1.1	Multinomial Naive Bayes	17
7.1.2	Support Vector Machine (Linear SVM)	17
7.1.3	Logistic Regression	18
7.2	Deep Learning Models	18
7.2.1	Convolutional Neural Network (CNN)	18
7.2.2	Long Short-Term Memory (LSTM / BiLSTM)	19
7.2.3	AraBERT	19
8	Experimental Setup	20
8.1	Datasets and label preparation	20
8.2	Preprocessing	20
8.3	Data splitting and validation	20
8.4	Feature extraction and feature selection	21
8.5	Models and hyperparameters	21
8.6	Software environment	21
9	Evaluation Metrics	22
9.1	Confusion Matrix	22
9.2	Accuracy	22
9.3	Precision	23
9.4	Recall	23
9.5	F1-score	23
9.6	ROC / AUC	23
10	Results and Discussion	24
10.1	Chapter Overview	24
10.2	Evaluation Metrics and Ranking	24
10.3	First Part: Full Parameter Comparison Results	24
10.3.1	Experimental Protocol	24
10.3.2	Best Configuration per Model Family	25
10.3.3	Top Results Within Each Dataset	25

10.3.4 Discussion of the Full Comparison	26
10.3.5 Limitations of the Full Comparison	27
10.4 Second Part: TF-IDF, FastText, and AraBERT Results	27
10.5 TF-IDF	27
10.5.1 Tweets Dataset	27
10.5.2 Reviews Dataset	32
10.6 FastText	35
10.6.1 Tweets Dataset	35
10.6.2 Reviews Dataset	38
10.7 AraBERT	41
10.7.1 Tweets Dataset	41
10.7.2 Reviews Dataset	43
10.8 Experimental Performance Synthesis	44
10.8.1 Binary Hotel Reviews	44
10.8.2 Binary Tweets	45
11 Challenges	46
12 Conclusion	47
13 Future Work	48

List of Figures

6.1	Feature selection comparison: cross-validated F_1 across methods (<code>none</code> , <code>chi2</code> , <code>mi</code> , <code>fisher</code>) with fixed extraction and classifier.	16
10.1	Dataset distribution and experimental protocol.	25
10.2	Ranked top configurations by macro F_1	26
10.3	Confusion Matrix for LinearSVC (4-class Tweet Dataset)	28
10.4	Comparison of F_1 -scores across Machine Learning baselines.	29
10.5	Confusion Matrix for the best performing model: Analysis of classification performance across sentiment categories.	30
10.6	Deep learning visualization.	31
10.7	Visual analysis of the initial dataset distribution.	33
10.8	Confusion Matrix for the top-performing LogReg model.	33
10.9	Visualization of Deep Learning Model Results	34
10.10	Visualization of Classical Model Comparisons	35
10.11	Confusion Matrix for LinearSVC	36
10.12	Performance Visualization of Deep Learning Models with FastText	37
10.13	Comparative analysis of classical models on hotel review data.	38
10.14	Confusion Matrix for Logistic Regression (Hotel Reviews)	39
10.15	Visualization of CNN vs. BiLSTM performance on hotel reviews.	40
10.16	Performance metrics for LinearSVC vs. Logistic Regression using AraBERT features on tweet data.	42
10.17	Confusion Matrix for LinearSVC with AraBERT (Tweet Dataset)	42
10.18	Performance metrics for LinearSVC vs. Logistic Regression using AraBERT features.	43
10.19	Confusion Matrix for LinearSVC with AraBERT Mean-Pooled Embeddings . . .	44

List of Tables

3.1	Overview of the two corpora (counts from the project files as analysed)	5
4.1	Qualitative summary of two sample transformations.	7
4.2	Raw vs. cleaned statistics (character length and whitespace-separated token counts).	8
5.1	Overview of representation families (typical use in this project).	9
10.1	Best configuration per section on ASTD binary sentiment	25
10.2	Best configuration per section on reviews_4star	25
10.3	Top five configurations by macro F_1 on ASTD	25
10.4	Top five configurations by macro F_1 on reviews_4star	26
10.5	Label Distribution	27
10.6	Comparative Performance of ML Models (Balanced 4-Way Classification)	28
10.7	Each ML model results)	29
10.8	Classification Report for the Best ML Model (LinearSVC)	30
10.9	Comparative Performance of Deep Learning Architectures	31
10.10	Classification Report for CNN Model	31
10.11	Classification Report for BiLSTM Model	32
10.12	Performance comparison of machine learning and deep learning models	32
10.13	Performance Comparison of Machine Learning Models (Binary Classification)	32
10.14	Deep Learning Model Performance Comparison	34
10.15	Classification Report for CNN Model	34
10.16	Classification Report for BiLSTM Model	34
10.17	Overall Model Performance Comparison (Binary Classification)	35
10.18	Machine Learning Model Performance Comparison	35
10.19	Classification Report for Best ML Model: LinearSVC	36
10.20	Deep Learning Model Performance Comparison (FastText Embeddings)	37
10.21	Classification Report: CNN Model (FastText)	37
10.22	Classification Report: BiLSTM Model (FastText)	37
10.23	Overall FastText performance on the tweet dataset	38
10.24	Performance Comparison of Classical Machine Learning Models	38
10.25	Classification Report for Best ML Model: LogReg	39
10.26	Deep Learning Model Performance Comparison (Hotel Reviews)	40
10.27	Classification Report for CNN Model	40
10.28	Classification Report for BiLSTM Model	40
10.29	Final Performance Summary: Deep Learning vs. Classical Baselines (Hotel Reviews)	41
10.30	Performance Comparison of Classical Heads on AraBERT Embeddings (Tweets)	41
10.31	Classification Report for Best Model (AraBERT): LinearSVC	42

10.32	Performance Comparison of Classical Heads on AraBERT Embeddings	43
10.33	Classification Report for Best Model (AraBERT): LinearSVC	44

Chapter 1

General Introduction

Sentiment Analysis, also known as opinion mining, is a Natural Language Processing (NLP) task that aims to determine the sentiment expressed in a text, such as positive, negative, or neutral opinions. With the rapid growth of social media and online platforms, large amounts of Arabic textual data are generated daily, making sentiment analysis increasingly important for understanding user opinions and behaviors.

However, Arabic sentiment analysis remains a challenging task due to the complexity of the Arabic language. These challenges include rich morphology, multiple dialects, ambiguous words, and informal writing styles commonly used in social media texts.

In this project, we study and compare different Machine Learning and Deep Learning approaches for Arabic sentiment classification using the HARD Arabic Reviews and ASTD datasets. Several text representation techniques, including Bag of Words (BoW), TF-IDF, FastText, and AraBERT, are explored. In addition, feature selection methods such as Chi-Square, Mutual Information, and Fisher Score are applied to improve classification performance.

The models are evaluated using metrics including Accuracy, Precision, Recall, F1-score, ROC/AUC, Confusion Matrix, and training time in order to identify the most effective approaches for Arabic sentiment analysis.

Chapter 2

Related Work

Sentiment Analysis has become one of the most active research areas in Natural Language Processing (NLP). Its objective is to identify opinions and emotions expressed in textual data. While sentiment analysis has achieved high performance in English, Arabic sentiment analysis remains more challenging due to the linguistic complexity of the Arabic language, including rich morphology, dialect diversity, and informal writing styles.

Several studies have explored Machine Learning (ML) approaches for Arabic sentiment classification. Traditional algorithms such as Naive Bayes, Support Vector Machines (SVM), and Logistic Regression are widely used because of their simplicity and effectiveness. These methods generally rely on feature extraction techniques such as Bag of Words (BoW) and TF-IDF to transform textual data into numerical representations. Among these models, SVM has often shown strong performance in text classification tasks due to its ability to handle high-dimensional data efficiently.

Researchers have also investigated the impact of preprocessing techniques on Arabic text classification. Common preprocessing steps include tokenization, normalization, stop-word removal, and stemming. These operations help reduce noise and improve the quality of textual features. However, preprocessing Arabic text remains difficult because of spelling variations, the absence of diacritics, and the existence of multiple dialects.

In recent years, Deep Learning (DL) methods have significantly improved sentiment analysis performance. Models such as Convolutional Neural Networks (CNN) and Long Short-Term Memory networks (LSTM) are capable of automatically learning semantic patterns from text data without requiring extensive manual feature engineering. LSTM models are particularly effective for sequential text processing because they can capture contextual dependencies between words.

More recently, transformer-based language models such as BERT have revolutionized NLP tasks. AraBERT, a pretrained language model specifically designed for Arabic, has achieved state-of-the-art results in several Arabic NLP applications, including sentiment analysis. Unlike traditional ML methods, AraBERT captures contextual and semantic information more effectively by using deep bidirectional representations of text.

In addition to representation techniques, feature selection methods have also been studied to improve classification efficiency and reduce dimensionality. Techniques such as Chi-Square, Mutual Information, and Fisher Score are commonly used to identify the most informative features and reduce computational complexity.

Despite the progress achieved in Arabic sentiment analysis, several challenges remain unresolved, particularly when dealing with dialectal Arabic and social media data. Therefore, comparing traditional ML approaches with modern DL methods remains an important research topic for understanding the strengths and limitations of each technique in Arabic sentiment classification.

Chapter 3

Dataset Description

This project uses two complementary Arabic corpora stored under `data/`: a microblog-style tweet collection (`Tweets.txt`) and a large tabular file of hotel guest reviews (`balanced-reviews.txt`). Together they cover *short, noisy social text* versus *longer, structured review text*, which is useful when comparing machine learning and deep learning models and when discussing domain shift.

3.1 Arabic tweets corpus (`data/Tweets.txt`)

3.1.1 Origin and format

The file `Tweets.txt` contains one record per line in the form `text\tLABEL`. The tweet appears first; the sentiment or stance label is separated by a tab and appears at the end of the line. All parsing and statistics below assume UTF-8 encoding. Blank lines are omitted; after cleaning empty rows, the corpus contains **10 006** labeled tweets.

3.1.2 Labels and interpretation

The scheme uses **four** labels (uppercase in file):

- **POS** — subjective positive sentiment (praise, congratulations, admiration).
- **NEG** — subjective negative sentiment (criticism, rejection, complaint).
- **NEUTRAL** — factual or mixed tone without a clear positive/negative stance toward a single target.
- **OBJ** — objective or informational content (news-style headlines, event announcements, invitations), often without evaluative opinion.

This is characteristic of Arabic Twitter datasets where “objective” and “neutral” are kept distinct: many posts are purely informational (**OBJ**), which differs from binary positive/negative movie or product reviews.

3.1.3 Class distribution and implications

The empirical distribution over the 10 006 tweets is:

Label	Count	Share (%)
OBJ	6 691	≈66.9
NEG	1 684	≈16.8
NEUTRAL	832	≈8.3
POS	799	≈8.0
Total	10 006	100

The corpus is **strongly imbalanced**: objective tweets dominate. Any model trained without adjustment may achieve high accuracy by predicting OBJ most of the time. For fair comparison, macro-averaged F1, per-class recall, or class-weighted training should be reported. The text itself is typical of Twitter: hashtags, mentions, code-switching, dialectal Egyptian and Levantine forms, elongation, and occasional non-Arabic tokens appear—so preprocessing (normalization, handling hashtags) materially affects results.

3.1.4 Typical content (qualitative)

Posts range from short reactions and political commentary to event invitations and news snippets. In the current file, tweet bodies average about **83** characters (maximum about 178 before the tab label). This dataset stresses *noise robustness* and *multi-class* learning rather than long-form argumentation.

3.2 Balanced hotel reviews (data/balanced-reviews.txt)

3.2.1 Origin and format

The file `balanced-reviews.txt` is **UTF-16 LE** (Unicode BOM FF FE); it must be read with the correct encoding when loading in Python or other tools. Each logical row is tab-separated with seven fields:

1. Row index (no),
2. Hotel name (Arabic),
3. Numerical **rating** on a 1–5 star scale,
4. Guest type (e.g. solo, couple, family),
5. Room type or “–”,
6. Stay duration phrase (e.g. “one night”),
7. Free-text **review** in Arabic (possibly with quotes and punctuation).

Using a robust CSV/TSV reader, **105 698** complete review rows are obtained (header excluded). The reviews refer to **1 173** distinct hotel names, i.e. a multi-property hospitality domain comparable in spirit to review datasets used in Arabic sentiment benchmarks (e.g. HARD-style accommodation feedback).

3.2.2 Star ratings

The star column is intended as a coarse sentiment proxy. In this file, only ratings **1**, **2**, **4**, and **5** occur; there is **no** rating 3 in the extracted rows (likely removed during balancing or filtering). Counts are:

Rating	Count	Share (%)
1 (worst)	14 382	≈13.6
2	38 467	≈36.4
4	26 450	≈25.0
5 (best)	26 399	≈25.0
Total	105 698	100

For binary sentiment experiments, a common mapping is: ratings 4–5 → positive, 1–2 → negative (rating 3 would be neutral if present). The intermediate rating 2 is frequent, so the corpus contains many *weakly negative* or mixed judgments expressed in full sentences.

3.2.3 Linguistic characteristics

Reviews are substantially **longer** than tweets: the review field averages about **137** characters per row in this file, with a long tail (maximum about 3 365 characters for a single review). Language is often Modern Standard Arabic mixed with Gulf or Levantine usage; guests describe cleanliness, staff, location, breakfast, and value. Compared to `Tweets.txt`, vocabulary is more repetitive (hotel-specific collocations) but spelling is usually more standard and less hashtag-driven.

3.3 Summary comparison

Table 3.1: Overview of the two corpora (counts from the project files as analysed)

Aspect	Tweets.txt	balanced-reviews.txt
Size	10 006 tweets	105 698 reviews
Labels	4-class: POS, NEG, NEUTRAL, OBJ	1–5 stars (1,2,4,5 present)
Encoding	UTF-8	UTF-16 LE
Domain	Social media, political/cultural mix	Hotel stays (multi-hotel)
Length	Short	Longer paragraphs
Balance	Highly skewed toward OBJ	Rating frequencies deliberately structured

These differences motivate separate preprocessing choices (e.g. hashtag handling for tweets, sentence segmentation for reviews) and careful interpretation of metrics when aggregating or comparing results across corpora.

Chapter 4

Data Preprocessing

Two corpora are cleaned with **different pipelines** implemented in `src/preprocessing.py`: `preprocess_tweet` for microblog text (`data/Tweets.txt`) and `preprocess_balanced_review` for hotel feedback (`data/balanced-reviews.txt`). Quantitative comparisons below were produced by `scripts/analyze_preprocessing.py` on the full files, with statistics saved as `report/preprocessing_stats.json`.

4.1 Pipeline for tweets (`preprocess_tweet`)

1. **Social tokens:** remove URLs; drop @mentions; replace hashtags `#word` by `word` so topical Arabic is retained; collapse long underscore/dash runs typical of noisy posts.
2. **Script filter:** remove Arabic diacritics not needed for polarity; keep Arabic letters and spaces; squeeze whitespace.
3. **Surface normalisation:** optional emoji→sentiment tokens; strip Latin punctuation; unify *alif/yaa/taa marbuta* variants (e.g. *alef* and *haa* forms).
4. **Deep clean:** repeated-character shortening (e.g. elongation), tatweel removal, non-Arabic symbol stripping while keeping underscore inside tokens when useful.

Stemming is **optional** (`use_stemming=True`); CSV export in `src/load_to_csv.py` enables it for archived tweet files.

4.1.1 Usage (code)

Listing 4.1 shows how the cleaners and binary star mapping are imported and called from application code or notebooks.

Listing 4.1: Import and single-document cleaning.

```
1 from src.preprocessing import (
2     preprocess_tweet,
3     preprocess_balanced_review,
4     preprocess_tweets,
5     preprocess_balanced_reviews,
6     star_rating_to_binary,
7 )
8
9 text_t = preprocess_tweet(raw_tweet_string)
10 text_r = preprocess_balanced_review(raw_review_string)
```

```

11
12 y_bin = star_rating_to_binary(row_rating) # "POS", "NEG", or None

```

Listing 4.2: Batch cleaning before vectorisation.

```

1 X_tweets_clean = preprocess_tweets(list_of_raw_tweets)
2 X_reviews_clean = preprocess_balanced_reviews(list_of_raw_review_cells)

```

4.2 Pipeline for hotel reviews (preprocess_balanced_review)

1. **Decorative quotes:** remove smart/ASCII quotes often pasted around review titles in scraped tables.
2. Same **script filter**, **emoji/punctuation/normalisation**, and **deep clean** as generic Arabic text (hashtag logic is disabled—reviews rarely use Twitter-style #).

4.3 Binary labels for star ratings

For experiments that need **two classes** on balanced-reviews.txt, numeric ratings are mapped with `star_rating_to_binary`: ratings 4–5 → POS, 1–2 → NEG; rating 3 can be dropped or folded (parameter `neutral`). The current extract contains only stars {1, 2, 4, 5}.

Listing 4.3: Binary mapping for the rating column (strings or ints).

```

1 label = star_rating_to_binary("5") # -> "POS"
2 label = star_rating_to_binary(1)   # -> "NEG"
3 label = star_rating_to_binary(3)   # -> None if neutral="drop" (default)

```

4.4 Illustrative text transformations

Real lines were taken from `data/Tweets.txt` and `data/balanced-reviews.txt` and processed with the project pipelines (see `scripts/sample_preprocessing_examples.py`). Table 4.1 summarises the *linguistic* effect; the exact UTF-8 strings are reproduced below.

Table 4.1: Qualitative summary of two sample transformations.

Corpus	What changes
Tweet	Hashtag markers (#) are removed while keeping the following Arabic letters as ordinary tokens; URLs and @mentions are removed; common orthographic rules merge alef variants and map <i>ta marbuta</i> to <i>haa</i> where configured.
Review	Decorative quotes and stray punctuation copied from booking tables are stripped before the same Arabic normalisation pass used for tweets (reviews do not use Twitter hashtag rules).

4.5 Empirical effect of cleaning (full-corpus statistics)

Cleaning shortens texts slightly by removing URLs, punctuation, and decorative symbols while preserving lexical content. Table 4.2 summarises length and token counts *before* (raw) and *after* (cleaned) preprocessing.

Table 4.2: Raw vs. cleaned statistics (character length and whitespace-separated token counts).

Corpus (<i>n</i>)	Characters		Tokens	
	Mean	Median	Mean	Median
Tweets raw (10 006)	83.6	86	14.2	14
Tweets cleaned	78.9	80	14.2	14
Reviews raw (105 698)	137.1	96	24.0	16
Reviews cleaned	130.0	91	23.2	16

Relative shortening: mean character length drops by about **6.4 %** for tweets and **7.0 %** for reviews (URLs, punctuation, and quote stripping). Token counts change modestly (≈ 0.3 tokens fewer per tweet and ≈ 0.8 per review on average), showing that the pipelines mainly *normalise surface form* rather than aggressively truncate content.

4.6 Discussion: differences between corpora after cleaning

- **Length:** cleaned reviews remain longer on average than tweets (≈ 130 vs. ≈ 79 characters mean), consistent with paragraph-style hotel feedback versus short posts.
- **Variance:** reviews show higher standard deviation in length (long outliers exist); tweets are more tightly clustered around a short length band.
- **Empty strings:** a handful of rows become empty after cleaning (5 tweets, 67 reviews in the measured run), mostly noise-only lines; these rows should be dropped before training.
- **Design implication:** hashtag-preserving cleaning suits opinion and topic cues in Twitter data; quote stripping suits tabular review exports. Using one generic cleaner for both domains would either lose hashtag semantics or skip hotel-specific noise removal.

4.7 Optional stemming

If enabled, a light Arabic stemmer (package `light_stemmer` when installed, else a rule-based fallback) reduces morphological variants to a common surface form; enable only when evaluating sparse bag-of-words models, since stemming can hurt neural models that rely on full word shapes.

Chapter 5

Text Representation

Machine learning and many deep models cannot consume raw strings: each document must be mapped to a fixed-length numerical representation (sparse counts, dense vectors, or contextual embeddings). This chapter summarises the representations used in Arabic sentiment analysis and relates them to the implementations in `src/Feature_Extraction.py` and `src/embeddings.py`.

5.1 Problem setting

Let $\mathcal{D} = \{d_1, \dots, d_N\}$ be a corpus of preprocessed Arabic texts. A vocabulary $\mathcal{V}$ of terms (words or n -grams) is built from training data. Each document d is mapped to a vector $\mathbf{x}(d) \in \mathbb{R}^{|\mathcal{V}|}$ (sparse) or $\mathbb{R}^k$ with $k \ll |\mathcal{V}|$ (dense), or to a sequence of token IDs for neural encoders.

Table 5.1: Overview of representation families (typical use in this project).

Family	Output shape / type	Role in the pipeline
BoW / TF-IDF	Sparse matrix ($N \times \mathcal{V} $)	Classical ML (NB, SVM, LR) with fast linear algebra
FastText / Ar-aVec	Dense word vectors averaged per doc	Bridging sparse lexicons and semantics
AraBERT	Contextual vectors per token / [CLS]	Strong baseline for Arabic NLP

5.2 Bag of Words (BoW)

5.2.1 Principle

The **bag-of-words** model ignores word order: each document is represented only by how often terms from $\mathcal{V}$ appear. After fixing an ordering of vocabulary terms $t_1, \dots, t_{|\mathcal{V}|}$, document d_i maps to

$$\mathbf{x}_i = [c_{i,1}, c_{i,2}, \dots, c_{i,|\mathcal{V}|}]^T,$$

where $c_{i,j}$ is the raw count (or binary presence) of term t_j in d_i . In matrix form, the whole corpus is a document-term matrix $\mathbf{X} \in \mathbb{R}^{N \times |\mathcal{V}|}$, usually very sparse.

5.2.2 *n*-grams and vocabulary control

Unigrams capture single tokens; bigrams and higher-order *n*-grams retain short local context (e.g. “not good” vs. “good”) at the cost of a larger $|\mathcal{V}|$. In this codebase, `CountVectorizer` supports `ngram_range=(1,2)` (unigrams + bigrams), `max_features` to cap $|\mathcal{V}|$, and `min_df` / `max_df` to drop overly rare or overly frequent terms. For Arabic, `lowercase=False` is appropriate because case folding is irrelevant and can harm Latin fragments.

5.2.3 Strengths and limits

BoW is fast and interpretable, and pairs well with linear models. It cannot capture long-range dependencies or polysemy: the same surface form has a single dimension regardless of context. Morphological richness in Arabic inflates vocabulary size unless stemming/normalisation is applied.

5.3 Term Frequency–Inverse Document Frequency (TF-IDF)

5.3.1 Motivation

Raw counts emphasise long documents and frequent function words. **TF-IDF** reweights terms so that words discriminative for a few documents receive higher weight than words that appear everywhere.

5.3.2 Components

Let $f_{t,d}$ be the raw frequency of term t in document d , and $\text{df}(t)$ the number of documents in which t appears. A standard weight is

$$\text{TF-IDF}(t, d) = w^{\text{TF}}(t, d) \times w^{\text{IDF}}(t), \quad (5.1)$$

with inverse document frequency often written as

$$w^{\text{IDF}}(t) = \log \frac{1 + N}{1 + \text{df}(t)} + 1 \quad (5.2)$$

(smoothed IDF as in `scikit-learn`), where N is the corpus size. Term frequency w^{TF} may be raw counts, normalised counts, or—as in this project—**sublinear TF**: $w^{\text{TF}} = 1 + \log f_{t,d}$ when $f_{t,d} > 0$, which dampens the influence of very frequent repetitions within one document (`sublinear_tf=True` in `TfidfVectorizer`).

5.3.3 Normalisation

The implementation applies **L2 normalisation** per row (`norm="l2"`), so each document vector lies on a hypersphere and cosine similarity equals the dot product—a common choice for text similarity and linear SVMs.

5.3.4 Relation to BoW

TF-IDF uses the same sparse indexing as BoW but replaces counts by TF-IDF weights. Feature selection (Chi-square, mutual information, etc.) can be applied on top of either representation.

5.4 Distributed representations: FastText and AraVec

5.4.1 Word embeddings

FastText (and comparable tools such as **AraVec**) learn dense vectors $\mathbf{e}_t \in \mathbb{R}^k$ for words (and subword units) from large unlabeled corpora using predictive objectives (skip-gram / CBOW). Semantically related words tend to have nearby vectors.

5.4.2 Subwords and Arabic

FastText represents each word as a sum of its character n -grams. That helps **rare words**, **typos**, and morphological variants common in dialectal or noisy Arabic: the model can partially share evidence across forms that share character sequences.

5.4.3 Document vectors

For classification, a simple recipe is to average word vectors of tokens in the document (possibly weighted by TF-IDF). This yields a dense $\mathbf{x}_i \in \mathbb{R}^k$ suitable for logistic regression or SVM with linear kernel. Quality depends on whether the embedding vocabulary covers the domain (hotel reviews vs. tweets).

5.5 Contextual representations: AraBERT

5.5.1 From words to transformers

Unlike static embeddings, **AraBERT** (BERT architecture trained on Arabic text) builds **contextual** vectors: the representation of a token depends on the full sentence. A classification head often uses the pooled representation of the special [CLS] token or a mean pool over subword pieces produced by the WordPiece tokenizer.

5.5.2 Fine-tuning

Starting from pretrained weights, **fine-tuning** updates all (or top) layers on the downstream sentiment dataset with a task-specific softmax layer. This typically yields the best accuracy among the representations discussed here, at higher computational cost (GPU recommended).

5.5.3 Arabic-specific benefit

General multilingual models underperform compared to Arabic-centric pretraining when dialect, morphology, and spelling variation matter; AraBERT is designed for Modern Standard Arabic and much user-generated Arabic text used in benchmarks.

5.6 Choosing a representation for this project

- **Baseline experiments:** TF-IDF + linear SVM / logistic regression is a strong, reproducible baseline (see `tfidf()` defaults: `ngram_range`, `max_features`, `min_df`, `max_df`).
- **Feature selection studies:** Chi-square / MI / Fisher scores apply naturally on sparse BoW or TF-IDF matrices with varying `max_features`.

- **Semantic gap:** FastText/AraVec averages help when exact word overlap between train and test is low.
- **Best accuracy budget:** Fine-tuned AraBERT is the usual ceiling model for Arabic sentiment, traded off against training time and inference cost.

Chapter 6

Feature Selection

High-dimensional sparse vectors from bag-of-words or TF—IDF can contain many weakly relevant dimensions. **Feature selection** keeps only the most informative dimensions (or keeps all as a baseline), which often *reduces training time and memory* and can *improve generalisation* when many noisy dimensions would otherwise dominate a linear decision boundary.

This project implements several classical selectors in `src/Feature_Selection.py` and compares them in a reproducible way via `src/ml_experiments.py` and the script `scripts/compare_feature`.

6.1 Notation

Let $\mathbf{X}_{\text{train}} \in \mathbb{R}^{n \times p}$ (typically sparse) be the matrix of vectorised training documents after a fixed feature extractor (e.g. TF—IDF), with binary labels $\mathbf{y} \in \{0, 1\}^n$. Feature selection chooses a subset of $k \ll p$ column indices so that both train and test matrices are restricted to the same columns, avoiding leakage by fitting scores *only* on the training fold.

6.2 Methods implemented

6.2.1 Baseline: no selection (none)

Idea: use the full vectorised vocabulary (subject to the extractor’s own `max_features`, `min_df`, etc.). This is the reference against which any selector must earn its keep.

Implementation: `select_features(..., method="none")` returns `X_train` and `X_test` unchanged (`Feature_Selection.py`).

6.2.2 Chi-square test (chi2)

Idea: score each dimension (term or n -gram) for **statistical dependence** with the class label under a χ^2 test of independence, treating non-negative feature values as categorical counts in the spirit of discrete tests. Terms whose counts co-vary strongly with positive vs. negative documents receive high scores.

Use case: standard, fast filter for sparse **non-negative** text features (raw counts or TF—IDF with non-negative entries).

Implementation: `SelectKBest` with `score_func=chi2`, keeping the top k terms (`Select_chi2`).

6.2.3 Mutual information (mi)

Idea: Mutual information (MI) measures how much knowing a feature reduces uncertainty about the label (in bits, up to a constant). It can detect non-linear associations that a linear correlation might miss.

Caveats: MI can be slower and more memory-hungry than χ^2 on very wide sparse matrices; scikit-learn may materialise dense blocks internally depending on configuration. It is still exposed as an optional filter for comparison.

Implementation: `SelectKBest` with `score_func=mutual_info_classif` (`Select_mi`).

6.2.4 Fisher score ranking (fisher)

Idea: for each dimension j , compare **between-class spread** to **within-class variance**. A common scalar score is

$$F_j = \frac{\sum_c n_c (\mu_{j,c} - \mu_j)^2}{\sum_c n_c \sigma_{j,c}^2 + \varepsilon}, \quad (6.1)$$

where $\mu_{j,c}$ and $\sigma_{j,c}^2$ are the mean and variance of feature j in class c , μ_j is the global mean of feature j , n_c is the class count, and ε avoids division by zero. Larger F_j suggests better class separation along dimension j .

Implementation: `Fisher_score` computes per-dimension scores on the training matrix (dense if input was sparse); `Select_fisher` keeps the indices of the top k dimensions and slices train and test consistently.

6.3 Selecting the best method: cross-validation over a discrete grid

We treat the **choice of selector** as a **categorical hyperparameter** with a small grid

$$\mathcal{M} = \{\text{none}, \text{chi2}, \text{mi}, \text{fisher}\}.$$

This is analogous in spirit to **grid search**: instead of optimising real-valued weights, we **search over finitely many preprocessing choices** and pick the one with the best validation score.

Important distinction: scikit-learn's `GridSearchCV` is used elsewhere in the project for **numeric** hyperparameters (e.g. SVM C , vectoriser `max_features`); the feature-selection comparison uses the same **evaluation protocol** (stratified K -fold on the training pool) but implements the loop explicitly so each fold **refits** the vectoriser, the selector, and the classifier only on training data—no test leakage.

Procedure (summary):

1. Split the corpus into a **training pool** and a **held-out test** set (stratified).
2. For each method $m \in \mathcal{M}$: run **stratified K -fold CV** on the training pool. In each fold, vectorise train/validation, apply selection m with fixed k , train a fixed classifier (e.g. linear SVM), record validation F_1 .
3. Choose $\arg \max_m$ of the **mean CV F_1** (tie-break with lower variance if desired).
4. **Refit** the winning m on the full training pool and report **hold-out** accuracy, precision, recall, and F_1 on the test set.

The helper `cross_val_score_pipeline` in `ml_experiments.py` encodes steps 2–3; `holdout_evaluate` supports step 4. The CLI script `scripts/compare_feature_selection.py` prints a sorted table and the best method automatically.

6.4 Code: running the comparison

Listing 6.1 shows the command-line entry point (run from the **repository root**). Listing 6.2 shows the equivalent programmatic pattern for a minimal notebook cell.

Listing 6.1: Shell: compare selectors with fixed TF—IDF and linear SVM (example).

```
1 python scripts/compare_feature_selection.py ^
2 --dataset reviews ^
3 --max-samples 12000 ^
4 --extraction tfidf ^
5 --methods none,chi2,mi,fisher ^
6 --k-select 5000 ^
7 --cv 5 ^
8 --classifier linear_svc ^
9 --output-csv report/feature_selection_cv.csv
```

Note: On Unix shells, replace ^ with line continuation \.

Listing 6.2: Python: same logic inside a notebook or driver script.

```
1 import sys
2 from pathlib import Path
3
4 ROOT = Path("..").resolve() # adjust if your cwd differs
5 sys.path.insert(0, str(ROOT / "src"))
6
7 from ml_experiments import (
8     init_rng,
9     load_binary_xy,
10    cross_val_score_pipeline,
11    holdout_evaluate,
12 )
13 from sklearn.model_selection import train_test_split
14
15 SEED = 42
16 init_rng(SEED)
17 X, y = load_binary_xy("reviews", max_samples=12000, random_state=SEED)
18 X_train, X_test, y_train, y_test = train_test_split(
19     X, y, test_size=0.2, random_state=SEED, stratify=y
20 )
21
22 mean_f1, std_f1 = cross_val_score_pipeline(
23     X_train, y_train,
24     extraction_method="tfidf",
25     selection_method="chi2",
26     k_select=5000,
27     classifier_name="linear_svc",
28     n_splits=5,
29     random_state=SEED,
30 )
31 print("chi2:", mean_f1, "+/-", std_f1)
```

6.5 Results

The feature-selection comparison is intended to show how the validation F_1 changes across `none`, `chi2`, `mi`, and `fisher` under a fixed representation and classifier. Figure 6.1 is kept as a visual slot for the exported comparison table or plot.

```
• Fitting 5 folds for each of 5 candidates, totalling 25 fits
Best params: {'clf__C': 0.5}
Accuracy: 0.9405
      precision    recall  f1-score   support

     0       0.9460      0.9343      0.9401       10564
     1       0.9351      0.9467      0.9409       10563

.934688 0.001577
bow      none 0.913660 0.002310

Best by CV mean_f1: tfidf
Hold-out: {'accuracy': 0.9335920859563591, 'f1': 0.9339609319839962, 'precision': 0.9287586594270736, 'recall': 0.9319852315}
```

Figure 6.1: Feature selection comparison: cross-validated F_1 across methods (`none`, `chi2`, `mi`, `fisher`) with fixed extraction and classifier.

6.6 Choosing k (number of retained features)

The hyperparameter k in `SelectKBest` / Fisher truncation should be chosen on **validation** data or inner CV. If k exceeds the number of columns produced by the vectoriser on a training fold, scikit-learn returns all available features and may emit a warning—reduce k or increase `max_features` in the extractor. Reporting a **curve** of F_1 vs. k (e.g. $k \in \{500, 1000, 2000, 5000\}$) is a useful extension beyond the single- k comparison table.

Chapter 7

Models

This chapter summarises the models used or considered in the project. The goal is not only to obtain a high score, but also to compare how classical sparse-feature classifiers and sequence-based deep models behave on Arabic tweets and hotel reviews.

7.1 Machine Learning Models

Classical machine learning models are trained on fixed-size vectors such as BoW or TF—IDF. In Arabic sentiment analysis, they are strong baselines because sentiment often depends on discriminative words and short phrases.

7.1.1 Multinomial Naive Bayes

Definition. Naive Bayes is a probabilistic classifier based on Bayes' rule and the assumption that features are conditionally independent given the class. For a document represented by features $\mathbf{x} = (x_1, \dots, x_p)$, the predicted class is

$$\hat{y} = \arg \max_c P(c) \prod_{j=1}^p P(x_j | c). \quad (7.1)$$

In practice, logarithms are used:

$$\hat{y} = \arg \max_c \left[\log P(c) + \sum_{j=1}^p x_j \log P(t_j | c) \right]. \quad (7.2)$$

Suitability for this project. Multinomial Naive Bayes is usually good as a fast baseline for BoW/TF—IDF text classification. It can perform well on short Arabic tweets when class-specific words are clear, but it may underperform linear SVM or logistic regression because the independence assumption is too simple for negation, dialectal expressions, and longer review sentences.

7.1.2 Support Vector Machine (Linear SVM)

Definition. A Support Vector Machine finds a separating hyperplane with the largest margin between classes. With a linear classifier, the prediction is

$$\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x} + b). \quad (7.3)$$

The soft-margin objective is commonly written as

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i, \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \quad (7.4)$$

The parameter C controls the trade-off between a wide margin and training errors.

Suitability for this project. Linear SVM is usually one of the strongest classical models for Arabic sentiment with TF—IDF, especially in high-dimensional sparse spaces. It is well suited to the project because it handles many features efficiently and often gives robust performance on both tweets and reviews.

7.1.3 Logistic Regression

Definition. Logistic regression is a linear classifier that estimates the probability of the positive class. For binary sentiment classification,

$$P(y = 1 | \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b) = \frac{1}{1 + \exp[-(\mathbf{w}^\top \mathbf{x} + b)]}. \quad (7.5)$$

The predicted class is positive when this probability exceeds a threshold, usually 0.5.

Suitability for this project. Logistic regression is a strong and interpretable baseline. It is usually competitive with linear SVM on TF—IDF features, and its learned weights can be inspected to identify which Arabic tokens or n -grams push predictions toward positive or negative sentiment.

7.2 Deep Learning Models

Deep learning models can consume token sequences and learn internal representations instead of relying only on manually fixed sparse features. They can model local word patterns, word order, and longer dependencies, but require more training time and benefit from GPU acceleration.

7.2.1 Convolutional Neural Network (CNN)

Definition. A CNN applies filters over consecutive token embeddings to detect local patterns similar to learned n -grams. If $\mathbf{e}_i$ is the embedding of token i , a convolutional filter of width h produces features such as

$$z_i = f(\mathbf{W}[\mathbf{e}_i; \mathbf{e}_{i+1}; \dots; \mathbf{e}_{i+h-1}] + b), \quad (7.6)$$

where f is a non-linear activation. A max-pooling layer then keeps the strongest detected pattern:

$$z_{\max} = \max_i z_i. \quad (7.7)$$

Suitability for this project. CNNs are often good for sentiment analysis because sentiment clues appear in short phrases. They are suitable for Arabic tweets and reviews when enough labeled data is available, but on smaller tweet subsets a TF—IDF + SVM baseline can still be stronger and more stable.

7.2.2 Long Short-Term Memory (LSTM / BiLSTM)

Definition. LSTM is a recurrent neural network designed to remember useful sequence information through gates. A simplified LSTM update is

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i), \quad (7.8)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f), \quad (7.9)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o), \quad (7.10)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \quad (7.11)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \quad (7.12)$$

A bidirectional LSTM (BiLSTM) reads the sequence from left to right and right to left, combining both contexts.

Suitability for this project. LSTM/BiLSTM models are useful for longer reviews where word order and negation can matter. However, they are more sensitive to dataset size, tokenisation, training epochs, and hardware. In this project they are useful as deep-learning baselines, but not guaranteed to outperform linear SVM on TF—IDF.

7.2.3 AraBERT

Definition. AraBERT is a transformer model pretrained on Arabic text. Transformers use self-attention, where each token representation attends to other tokens in the sequence. A simplified attention operation is

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \quad (7.13)$$

For classification, the model representation (often the [CLS] token or pooled output) is passed to a classification layer:

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{W}\mathbf{h}_{[\text{CLS}]} + \mathbf{b}). \quad (7.14)$$

Suitability for this project. AraBERT is usually very good for Arabic sentiment analysis because it starts from language knowledge learned from large Arabic corpora. It is expected to outperform simple sparse-feature models when fine-tuned correctly, but it requires more memory, time, and preferably a GPU. For this project, AraBERT is best treated as an advanced model or future improvement if compute resources are limited.

Chapter 8

Experimental Setup

This chapter describes the practical setup used to run the experiments in the project. The same general protocol is followed for both datasets so that model comparisons are fair.

8.1 Datasets and label preparation

Two datasets are used:

- **Tweets dataset:** `ASTD/data/Tweets.txt`, originally labelled with POS, NEG, NEUTRAL, and OBJ. For binary experiments, only POS and NEG are kept and balanced.
- **Hotel reviews dataset:** `data/balanced-reviews.txt`, loaded with UTF-16 encoding. Star ratings are mapped as 4–5 → POS and 1–2 → NEG; rating 3 is dropped if present.

8.2 Preprocessing

Preprocessing is implemented in `src/preprocessing.py`. Tweets use a tweet-specific pipeline that handles URLs, mentions, hashtags, Arabic normalisation, elongation, punctuation, and optional emoji sentiment tokens. Reviews use a review-specific pipeline that additionally removes decorative quotes copied from table exports. Empty cleaned texts are removed before training.

8.3 Data splitting and validation

The standard split is:

- **80% training pool** and **20% held-out test set**;
- **stratification** is used so positive and negative classes remain balanced in each split;
- method selection and hyperparameter search are performed using cross-validation on the training pool only;
- the held-out test set is used once for final reporting.

This avoids information leakage from the test set into feature selection or hyperparameter tuning.

8.4 Feature extraction and feature selection

Sparse text features are produced with:

- **BoW** using `CountVectorizer`;
- **TF—IDF** using `TfidfVectorizer`, usually with unigrams and bigrams, `lowercase=False`, `min_df`, `max_df`, and `max_features`.

Feature selection methods (`none`, `chi2`, `mi`, `fisher`) are compared with fixed extraction and classifier settings using `scripts/compare_feature_selection.py`. Feature extraction methods are compared with `scripts/compare_feature_extraction.py`.

8.5 Models and hyperparameters

The classical models are **Multinomial Naive Bayes**, **Linear SVM**, and **Logistic Regression**. Hyperparameters such as SVM C , TF—IDF vocabulary size, n -gram range, and document-frequency thresholds are tuned with `scripts/gridsearch_hyperparams.py`. Deep-learning experiments use Keras/TensorFlow CNN and BiLSTM notebooks with token sequences, embedding layers, a fixed maximum sequence length, and a small number of epochs for comparison.

8.6 Software environment

The project uses Python with `pandas`, `numpy`, `scikit-learn`, `matplotlib`, and `TensorFlow/Keras` for deep learning. Experiments are organised in reusable modules under `src/` and command-line scripts under `scripts/`, while notebooks are kept mostly as readable experiment reports.

Chapter 9

Evaluation Metrics

Evaluation metrics convert predictions into interpretable numbers. In binary sentiment classification, each prediction belongs to one of four cases:

- **TP** (true positive): a positive text predicted as positive;
- **TN** (true negative): a negative text predicted as negative;
- **FP** (false positive): a negative text predicted as positive;
- **FN** (false negative): a positive text predicted as negative.

9.1 Confusion Matrix

A **confusion matrix** is a table that counts correct and incorrect predictions for each class. For binary sentiment, using **NEG** as class 0 and **POS** as class 1, the matrix is

	Predicted NEG	Predicted POS
Actual NEG	<i>TN</i>	<i>FP</i>
Actual POS	<i>FN</i>	<i>TP</i>

This matrix is the starting point for accuracy, precision, recall, and F_1 . It is also useful visually: a good model has large values on the diagonal (*TN*, *TP*) and small values off the diagonal (*FP*, *FN*).

9.2 Accuracy

Accuracy measures the proportion of all predictions that are correct:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

It is easy to understand, but it can be misleading if classes are imbalanced. For this reason, F_1 is also reported.

9.3 Precision

Precision answers: among the texts predicted as positive, how many were truly positive?

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

High precision means few negative texts are incorrectly labelled as positive.

9.4 Recall

Recall answers: among all true positive texts, how many did the model find?

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

High recall means the model misses few positive examples.

9.5 F1-score

The F_1 -score is the harmonic mean of precision and recall:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

It is especially useful in this project because it balances false positives and false negatives better than accuracy alone.

9.6 ROC / AUC

ROC/AUC can be used when a model outputs scores or probabilities. The ROC curve plots true positive rate against false positive rate at different thresholds, and AUC summarises the curve in one value. In this project, AUC is optional because some strong models such as `LinearSVC` naturally output margins rather than calibrated probabilities.

Chapter 10

Results and Discussion

10.1 Chapter Overview

This chapter is divided into two complementary parts. The first part reports the full parameter-comparison experiment from `notebooks/full_parameter_comparison_report_copy.ipynb`, using the synthesis in `comparison_results_preview.md` for interpretation. The second part keeps the previously added detailed results for TF-IDF, FastText, and AraBERT.

The two datasets are discussed under separate protocols: ASTD is a balanced binary tweet-sentiment task, while `reviews_4star` is a four-class hotel-rating task. Their scores should therefore be interpreted within each dataset, not as a single global ranking.

10.2 Evaluation Metrics and Ranking

Before detailing the results, it is important to note the following evaluation criteria:

- **Ranking metric:** Models are ordered mainly by macro F_1 , which averages class-level performance and is more reliable than accuracy when classes are imbalanced or multi-class.
- **Accuracy:** Accuracy is reported as a secondary metric because it is easy to interpret, but it can hide weak recall on minority classes.
- **ROC-AUC:** For ASTD, ROC-AUC is computed on the positive class. For `reviews_4star`, one-vs-rest macro ROC-AUC is used.

10.3 First Part: Full Parameter Comparison Results

10.3.1 Experimental Protocol

The full comparison evaluates 110 completed configurations with a fixed random seed (`SEED = 42`) and a stratified 80/20 train-test split. The experiment compares four families of approaches:

- sparse machine learning with BoW or TF-IDF, feature selection, and linear classifiers;
- shallow Keras neural models using token sequences, mainly CNN and LSTM;

- FastText/AraVec document vectors combined with feature selection and linear classifiers;
- end-to-end fine-tuned AraBERT.

For ASTD, only the POS and NEG labels are retained and balanced, giving 1,597 documents (1,277 train / 320 test). For reviews_4star, the retained ratings are 1, 2, 4, and 5 stars, giving 105,698 reviews after filtering (84,504 train / 21,127 test).

```
ASTD | task=binary | n=1597 | train 1277 | test 320
NEG: 799 | POS: 798
reviews_4star | task=multiclass | n=105631 | train 84504 | test 21127
★1: 14378
★2: 38441
★4: 26441
★5: 26371
```

Figure 10.1: Dataset distribution and experimental protocol.

10.3.2 Best Configuration per Model Family

Table 10.1: Best configuration per section on ASTD binary sentiment

Section	F1 macro	Accuracy	ROC-AUC	Configuration
AraBERT fine-tune	0.815	0.816	0.903	End-to-end BERT, 2 epochs
FastText + ML	0.794	0.794	0.856	AraVec + χ^2 ($K = 300$) + LR
ML sparse	0.716	0.716	0.740	TF-IDF + MI ($K = 400$) + LinearSVC
DL Keras	0.647	0.647	0.689	CNN, vocab 400

Table 10.2: Best configuration per section on reviews_4star

Section	F1 macro	Accuracy	ROC-AUC	Configuration
AraBERT fine-tune	0.800	0.817	0.961	End-to-end BERT, 2 epochs
DL Keras	0.755	0.781	0.946	CNN, vocab 1000
ML sparse	0.736	0.760	0.933	BoW + Fisher ($K = 400$) + LR
FastText + ML	0.660	0.704	0.939	AraVec + Fisher ($K = 300$) + LR

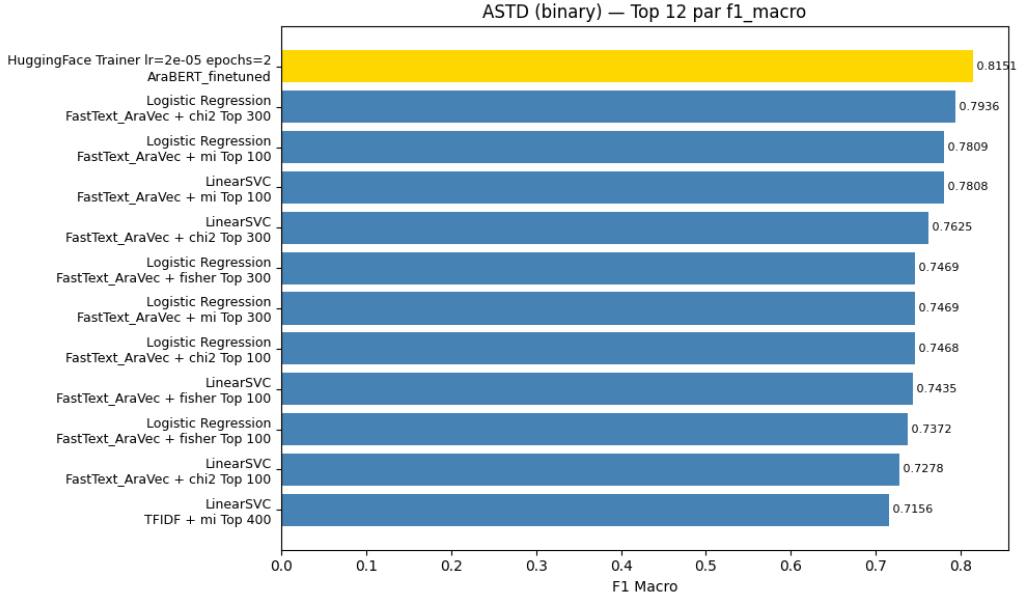
10.3.3 Top Results Within Each Dataset

Table 10.3: Top five configurations by macro F_1 on ASTD

F1	Accuracy	Section	Model pipeline
0.815	0.816	AraBERT fine-tune	Fine-tuned AraBERT
0.794	0.794	ML FastText	AraVec + χ^2 (300) + LR
0.781	0.781	ML FastText	AraVec + MI (100) + LR
0.762	0.763	ML FastText	AraVec + χ^2 (300) + LinearSVC
0.716	0.716	ML sparse	TF-IDF + MI (400) + LinearSVC

Table 10.4: Top five configurations by macro F_1 on reviews_4star

F1	Accuracy	Section	Model pipeline
0.800	0.817	AraBERT fine-tune	Fine-tuned AraBERT
0.755	0.781	DL Keras	CNN, vocab 1000
0.746	0.776	DL Keras	CNN, vocab 400
0.736	0.760	ML sparse	BoW + Fisher (400) + LR
0.735	0.761	ML sparse	TF-IDF + χ^2 (400) + LR

Figure 10.2: Ranked top configurations by macro F_1 .

10.3.4 Discussion of the Full Comparison

Fine-tuned AraBERT is the strongest approach in both protocols, reaching 0.815 macro F_1 on ASTD and 0.800 macro F_1 on reviews_4star. This confirms the value of contextual Arabic pre-training: the model captures negation, polysemy, morphology, and domain phrasing better than static or sparse representations. Even with only two fine-tuning epochs, it adapts well enough to outperform the other model families.

The second important result is the strong performance of AraVec/FastText on tweets. On ASTD, AraVec with feature selection and logistic regression reaches approximately 0.79 macro F_1 , clearly above sparse TF-IDF/BoW baselines. Tweets are short and noisy, so subword information helps recover useful sentiment cues from spelling variation, dialectal words, and rare forms.

The same FastText strategy is weaker on hotel reviews, where it reaches only about 0.66 macro F_1 . Averaging word vectors over long reviews smooths local sentiment signals, especially when a review mixes praise and criticism. In this setting, sparse BoW/TF-IDF features remain competitive because high-dimensional n -grams preserve rating-specific phrases and collocations.

Sparse machine-learning models remain useful baselines. Increasing the selected feature count from small values to $K = 400$ improves lexical coverage, and the best feature-selection method depends on the corpus. TF-IDF with mutual information is strongest for ASTD, while BoW with Fisher scoring is strongest for reviews_4star.

The shallow neural models show mixed behavior. The CNN performs better than the LSTM in the reported tweet setup, while the LSTM collapses on ASTD to a near single-class predictor, producing macro $F_1 \approx 0.33$ and accuracy near 0.50. On the larger review corpus, neural models improve substantially because the dataset provides enough examples to train sequence patterns, with the best CNN reaching 0.755 macro F_1 .

10.3.5 Limitations of the Full Comparison

The comparison uses a reduced grid rather than exhaustive hyperparameter tuning. Frozen AraBERT embeddings are not included in the merged result table, and the computational cost is uneven: sparse models train in seconds, while AraBERT fine-tuning on the review corpus is much more expensive. Finally, the two datasets should remain analytically separate because one is balanced binary sentiment and the other is four-class rating prediction.

10.4 Second Part: TF-IDF, FastText, and AraBERT Results

The following sections keep the previously added report results and discussion for TF-IDF, FastText, and AraBERT.

10.5 TF-IDF

10.5.1 Tweets Dataset

Class Distribution and Dataset Balancing

Initial exploratory analysis revealed a significant class imbalance, with the OBJ label dominating the distribution. This skew negatively impacted preliminary results, leading to sub-optimal model performance.

Table 10.5: Label Distribution

Label	Original Count
OBJ	6,691
NEG	1,684
NEUTRAL	832
POS	799
Total	10,006

Machine Learning Baseline Results

Following the vectorization of text using TF-IDF (with n -grams up to 2), we evaluated three classical classifiers. The models were ranked based on their Macro F_1 -score to account for performance across all four classes equally.

The 4×4 confusion matrix for LinearSVC reveals modest class separation, consistent with the recorded $\sim 43\%$ accuracy. The results highlight two primary challenges:

- **Lexical Overlap:** NEUTRAL and OBJ categories show substantial off-diagonal confusion due to a lack of distinct sentiment markers.

Table 10.6: Comparative Performance of ML Models (Balanced 4-Way Classification)

Model	Accuracy	Macro F_1 -Score
LinearSVC	0.4320	0.4315
LogReg	0.4150	0.4102
MNB	0.3980	0.3854

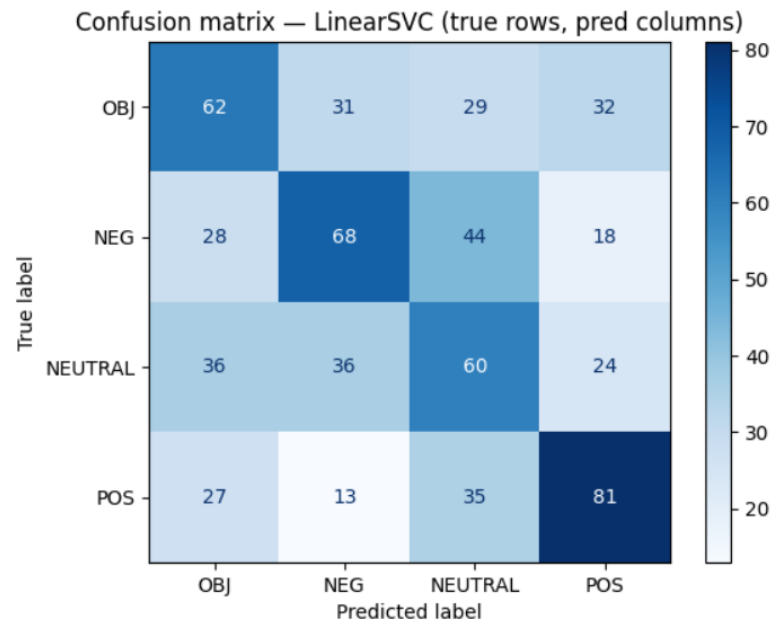


Figure 10.3: Confusion Matrix for LinearSVC (4-class Tweet Dataset)

- **Representation Limits:** Shallow TF-IDF features struggle to distinguish objective descriptions from neutral sentiment.

Because OBJ and NEUTRAL classes introduce significant noise and overlap, the remainder of this analysis will focus exclusively on the POS and NEG classes, which represent the primary intended targets for sentiment classification.

ML models Results comparison

Table 10.7: Each ML model results)

Model	Accuracy	F1
SVM	69%	0.68
NB	70%	0.68
LogReg	68%	0.67

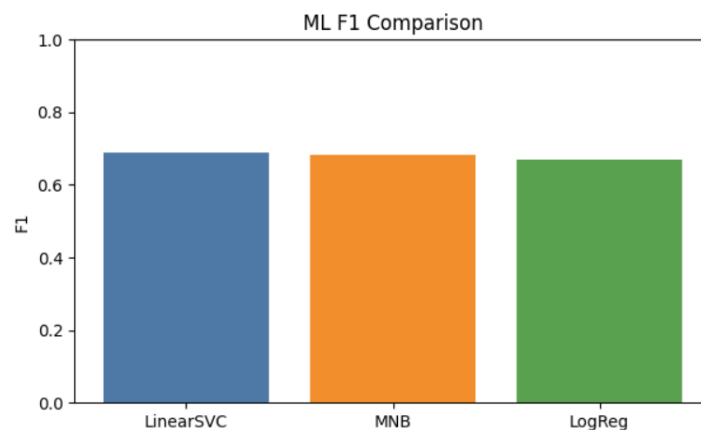


Figure 10.4: Comparison of F_1 -scores across Machine Learning baselines.

Discussion: Analyzing the Classical Baseline Ceiling

Despite the results clustering closely, the classical models share a performance ceiling due to their reliance on surface-level word patterns via TF-IDF. This approach presents significant challenges in the context of Arabic Natural Language Processing, specifically regarding:

- **Morphological Complexity:** The intricate root-and-pattern system and the use of clitics.
- **Dialectal Variation:** The inherent noise found in informal or regional speech patterns.
- **Preprocessing Trade-offs:** While normalization and diacritic stripping simplify the learning process, they remove granular data that richer models could exploit.

Nevertheless, we will now focus specifically on LinearSVC, as it emerged as the most robust model within this cluster.

Table 10.8: Classification Report for the Best ML Model (LinearSVC)

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.6918	0.6962	0.6940	158
1 (Positive)	0.6903	0.6859	0.6881	156
Accuracy		0.6911		314
Macro Avg	0.6911	0.6910	0.6911	314
Weighted Avg	0.6911	0.6911	0.6911	314

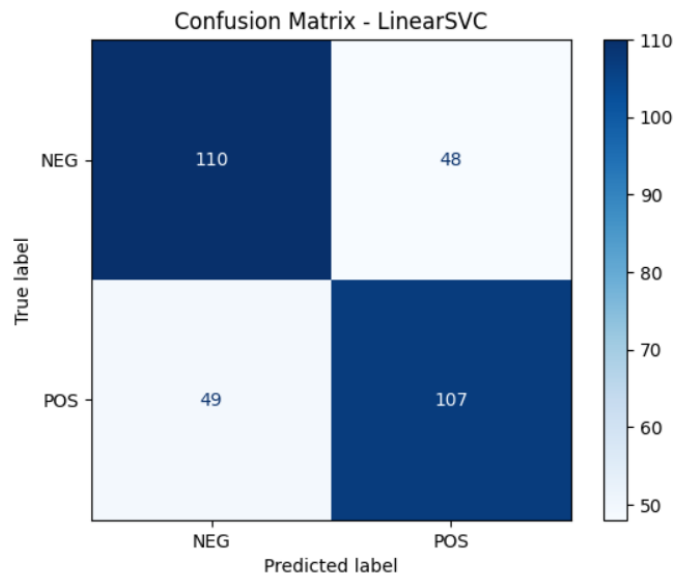


Figure 10.5: Confusion Matrix for the best performing model: Analysis of classification performance across sentiment categories.

The confusion matrix indicates a moderate, class-balanced separability under the TF-IDF and LinearSVC framework. The results demonstrate the following classification behaviors:

- **Balanced Error Distribution:** Failure modes are split evenly between false negatives (missing positives) and false positives (over-predicting positives).
- **Performance Consistency:** This symmetry is consistent with the nearly identical F_1 -scores recorded for the NEG and POS classes.
- **Classification Accuracy:** Most tweets are classified correctly, validating the effectiveness of the linear SVM approach for binary sentiment detection.

This balanced performance suggests that the model does not exhibit a significant bias toward either sentiment.

On sparse, high-dimensional n -gram vectors, a linear decision boundary is generally appropriate.

Consequently, the ranking reflects a modest empirical edge for these specific settings and data splits, rather than a definitive superiority of SVM in all contexts.

Deep Learning Models: Results & Comparison

Table 10.9: Comparative Performance of Deep Learning Architectures

Model	Accuracy	F1
CNN	69%	0.68
BiLSTM	60%	0.61

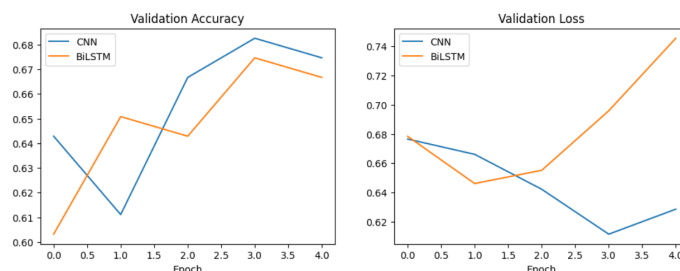


Figure 10.6: Deep learning visualization.

Table 10.10: Classification Report for CNN Model

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.6918	0.6962	0.6940	158
1 (Positive)	0.6903	0.6859	0.6881	156
Accuracy		0.6911		314
Macro Avg	0.6911	0.6910	0.6911	314
Weighted Avg	0.6911	0.6911	0.6911	314

Table 10.11: Classification Report for BiLSTM Model

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.6174	0.5823	0.5993	158
1 (Positive)	0.6000	0.6346	0.6168	156
Accuracy		0.6083		314
Macro Avg	0.6087	0.6084	0.6081	314
Weighted Avg	0.6088	0.6083	0.6080	314

Comparison between the deep learning architectures highlights a performance trade-off:

- **CNN (Balanced Baseline):** Higher overall accuracy and a stable error distribution, making it the more reliable model for this split.
- **BiLSTM (Aggressive Prediction):** Higher POS F_1 , but at the cost of misclassifying many negative tweets as positive.
- **Impact:** The BiLSTM's bias degrades NEG recall and overall performance compared to the more consistent CNN.

Overall comparison

Evaluation of the experimental results suggests the following conclusions regarding the model architectures:

Model	Accuracy	F1-score
LinearSVC	0.691083	0.688103
CNN	0.691083	0.688103
MNB	0.707006	0.682759
LogReg	0.681529	0.671053
BiLSTM	0.608280	0.616822

Table 10.12: Performance comparison of machine learning and deep learning models

- **TF-IDF + LinearSVC:** Provides the best all-around trade-off between computational efficiency and classification performance.
- **CNN:** Serves as a highly competitive sequence-based baseline, maintaining balanced metrics.
- **BiLSTM:** Optimizes specifically for positive sentiment, but at the cost of high false-positive rates for negative tweets.

10.5.2 Reviews Dataset

ML models Results comparison

Table 10.13: Performance Comparison of Machine Learning Models (Binary Classification)

Model	Accuracy	F1-score
LogReg	0.9421	0.9425
LinearSVC	0.9329	0.9332
MNB	0.9167	0.9160

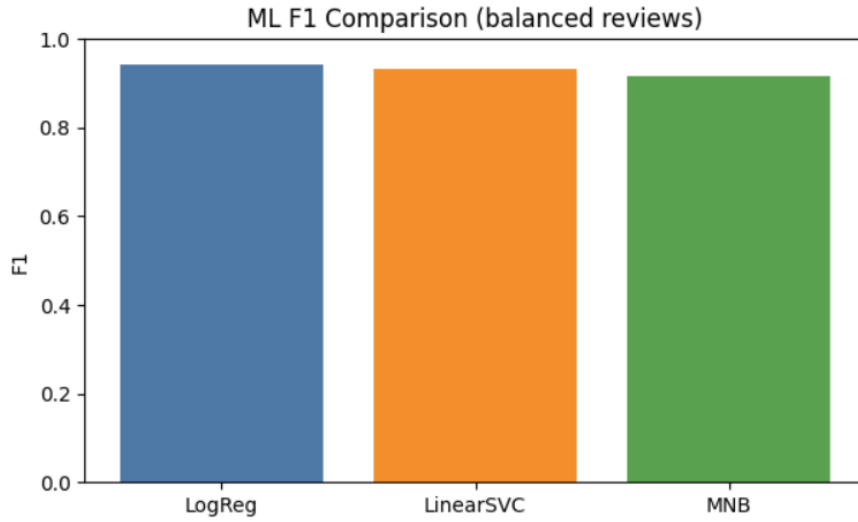


Figure 10.7: Visual analysis of the initial dataset distribution.

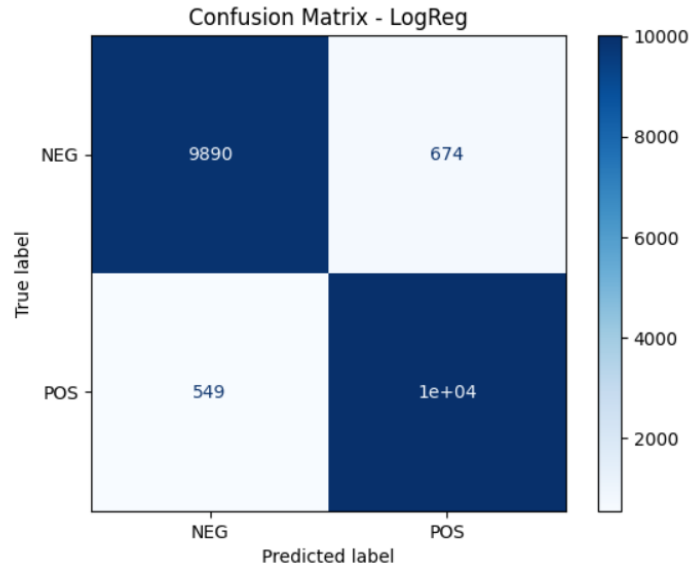


Figure 10.8: Confusion Matrix for the top-performing LogReg model.

The confusion matrix reveals balanced error rates between classes, with misclassifications split nearly evenly between negative and positive reviews.

This symmetry confirms that the model's accuracy is driven by consistent performance across both polarities rather than a systematic bias.

The selection of the optimal classical model follows the same methodology as the Arabic tweet analysis. Since all three models utilize identical TF-IDF n -gram features, LogReg was selected based on its superior F_1 -score on the positive class.

This confirms that for sparse, high-dimensional bag-of-words signals, the logistic regression objective provides the most effective decision boundary for this specific data split.

Deep Learning Models: Results & Comparison

Table 10.14: Deep Learning Model Performance Comparison

Model	Accuracy	F1-score
BiLSTM	0.9384	0.9381
CNN	0.9275	0.9274

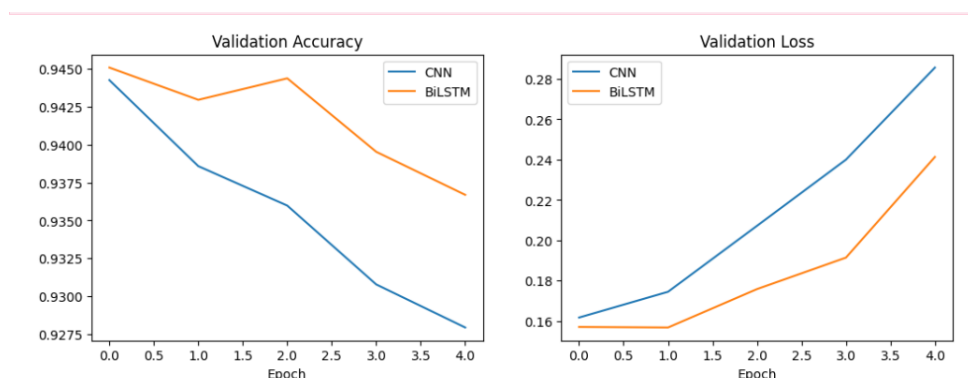


Figure 10.9: Visualization of Deep Learning Model Results

Table 10.15: Classification Report for CNN Model

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.9264	0.9288	0.9276	10,564
1 (Positive)	0.9286	0.9262	0.9274	10,563
Accuracy		0.9275		21,127
Macro Avg	0.9275	0.9275	0.9275	21,127
Weighted Avg	0.9275	0.9275	0.9275	21,127

Table 10.16: Classification Report for BiLSTM Model

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.9342	0.9433	0.9387	10,564
1 (Positive)	0.9427	0.9335	0.9381	10,563
Accuracy		0.9384		21,127
Macro Avg	0.9385	0.9384	0.9384	21,127
Weighted Avg	0.9385	0.9384	0.9384	21,127

For the CNN model, training accuracy continues to climb while the validation loss begins to worsen over successive epochs. This diverging trend is a clear indicator of overfitting, where the model fits the training slice increasingly well without matching gains on the validation set.

In contrast, the BiLSTM maintains validation accuracy within a tighter band (~0.93–0.94) and demonstrates a smaller gap relative to its training behavior.

This stability directly correlates with the superior test results observed after thresholding at 0.5, suggesting better generalization for this specific sequence-based task.

Overall comparison

Table 10.17: Overall Model Performance Comparison (Binary Classification)

Model	Accuracy	F1-score
LogReg	0.9421	0.9425
BiLSTM	0.9384	0.9381
LinearSVC	0.9329	0.9332
CNN	0.9275	0.9274
MNB	0.9167	0.9160

On Arabic hotel reviews, Logistic Regression using TF-IDF (1,2)-grams achieves the highest test metrics among all baselines.

This indicates that lexical n -gram signals remain highly predictive under this configuration, with the short-trained CNN and BiLSTM architectures failing to surpass the classical baseline within the chosen experimental setup.

10.6 FastText

10.6.1 Tweets Dataset

ML models Results comparison

Table 10.18: Machine Learning Model Performance Comparison

Model	Accuracy	F1-score
LinearSVC	0.7771	0.7682
LogReg	0.7548	0.7492

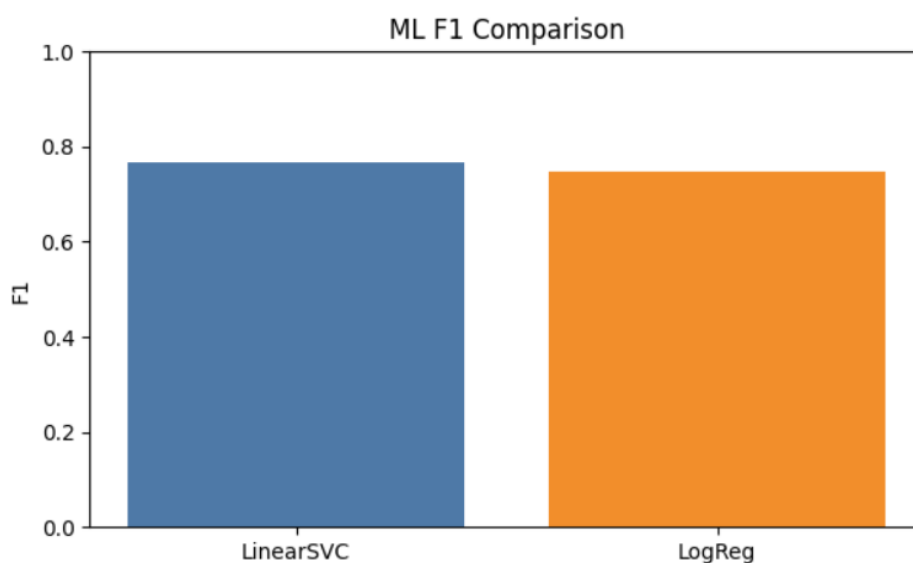


Figure 10.10: Visualization of Classical Model Comparisons

Table 10.19: Classification Report for Best ML Model: LinearSVC

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.7619	0.8101	0.7853	158
1 (Positive)	0.7945	0.7436	0.7682	156
Accuracy		0.7771		314
Macro Avg	0.7782	0.7769	0.7767	314
Weighted Avg	0.7781	0.7771	0.7768	314

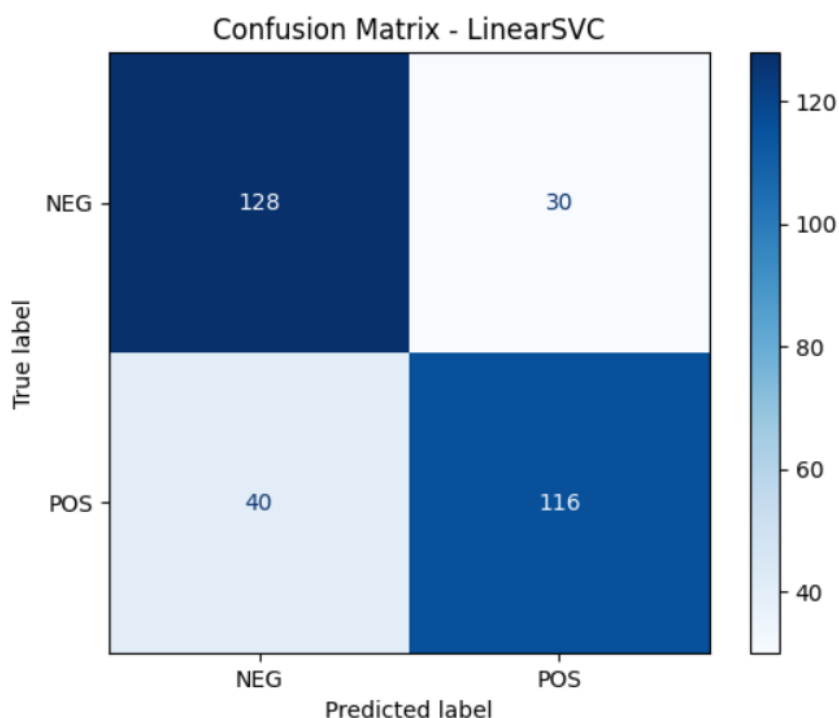


Figure 10.11: Confusion Matrix for LinearSVC

Under identical L_2 -normalized FastText vectors, LinearSVC slightly outperformed Logistic Regression.

This advantage stems from the SVM objective's focus on ambiguous examples, which is effective given that FastText embeddings rarely provide a clean separation between sentiment classes in vector space.

The resulting performance favors negative recall over positive recall.

With approximately 40 false negatives for positive tweets compared to 30 false positives for negative tweets across 314 examples, the results indicate that positive sentiment is moderately more difficult to detect under this frozen embedding and linear setup.

Deep Learning Models: Results & Comparison

Table 10.20: Deep Learning Model Performance Comparison (FastText Embeddings)

Model	Accuracy	F1-score
CNN	0.6815	0.6711
BiLSTM	0.6688	0.6923

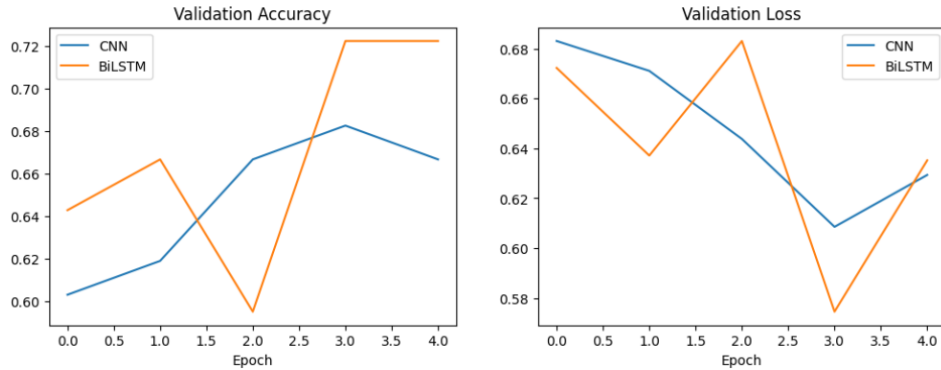


Figure 10.12: Performance Visualization of Deep Learning Models with FastText

Table 10.21: Classification Report: CNN Model (FastText)

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.6747	0.7089	0.6914	158
1 (Positive)	0.6892	0.6538	0.6711	156
Accuracy		0.6815		314
Macro Avg	0.6819	0.6814	0.6812	314
Weighted Avg	0.6819	0.6815	0.6813	314

Table 10.22: Classification Report: BiLSTM Model (FastText)

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.7045	0.5886	0.6414	158
1 (Positive)	0.6429	0.7500	0.6923	156
Accuracy		0.6688		314
Macro Avg	0.6737	0.6693	0.6668	314
Weighted Avg	0.6739	0.6688	0.6667	314

On the balanced Arabic tweet dataset using 128-dimensional learned embeddings, a performance trade-off exists between architectures: the BiLSTM achieves a higher positive-class F_1 (≈ 0.69) by capturing global sequential dependencies and word order, whereas the CNN maintains higher overall accuracy (≈ 0.68) by prioritizing local salient phrases via GlobalMaxPooling.

This behavior reflects a metric mismatch where the BiLSTM's recall shift toward the positive class improves F_1 even as total accuracy dips.

Overall comparison

Table 10.23: Overall FastText performance on the tweet dataset

Model	Accuracy	F1-score
LinearSVC	0.7771	0.7682
LogReg	0.7548	0.7492
CNN	0.6815	0.6711
BiLSTM	0.6688	0.6923

Evaluation across architectures indicates that classical machine learning using AraVec (FastText) sentence vectors remains the superior approach for this dataset.

LinearSVC achieves approximately 0.78 accuracy and 0.77 positive-class F_1 , marginally outperforming Logistic Regression on identical features.

In contrast, the neural architectures CNN and BiLSTM trained on scratch-learned embeddings reach only 0.67–0.68 accuracy. While the BiLSTM yields a higher positive F_1 (~ 0.69) than the CNN, validation curves reveal limited generalization.

Conclusion: The FastText-based linear models dominate the neural baselines in this setup.

10.6.2 Reviews Dataset

ML models Results comparison

Table 10.24: Performance Comparison of Classical Machine Learning Models

Model	Accuracy	F1-score
LogReg	0.9066	0.9063
LinearSVC	0.9064	0.9060

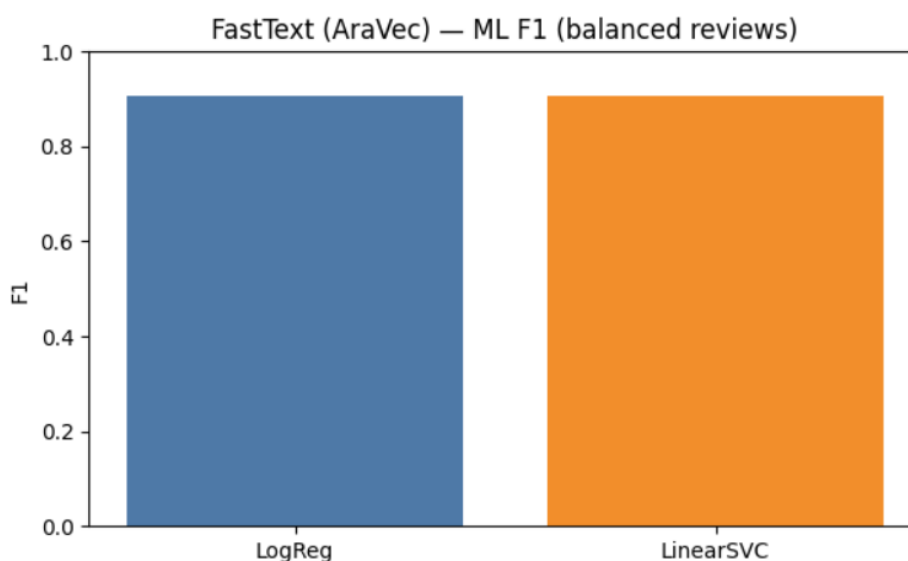


Figure 10.13: Comparative analysis of classical models on hotel review data.

Table 10.25: Classification Report for Best ML Model: LogReg

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.9043	0.9094	0.9068	10,564
1 (Positive)	0.9089	0.9037	0.9063	10,563
Accuracy		0.9066		21,127
Macro Avg	0.9066	0.9066	0.9066	21,127
Weighted Avg	0.9066	0.9066	0.9066	21,127

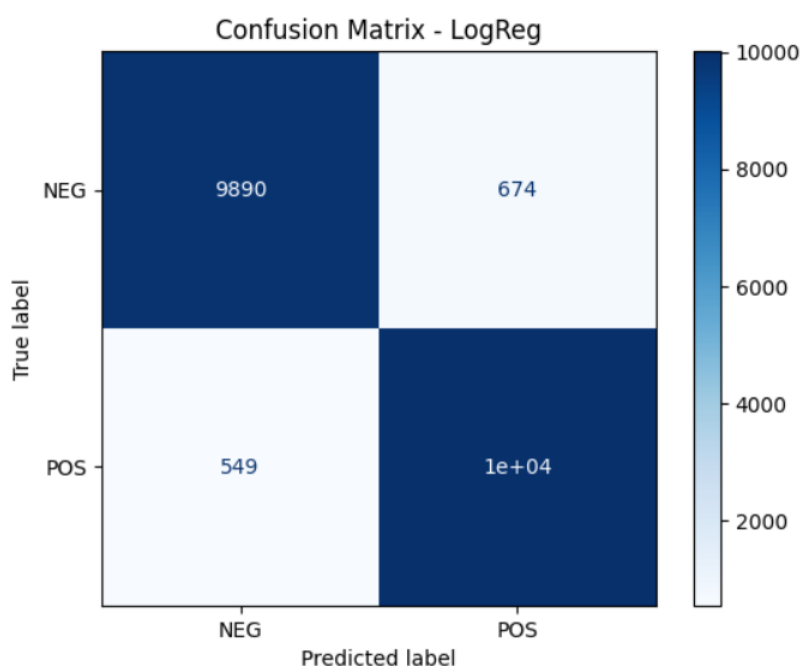


Figure 10.14: Confusion Matrix for Logistic Regression (Hotel Reviews)

LogReg performs optimally only by a marginal difference; since both linear models process identical bag-of-token-average `FastText` vectors, their performance is nearly indistinguishable.

In this instance, LogReg achieves slightly higher overall accuracy and positive-class F_1 while maintaining a confusion structure nearly identical to LinearSVC.

Errors remain balanced across classes, with approximately 1.9k misclassifications out of 21k total samples.

The error distribution is highly symmetric:

- **False Negatives (POS → NEG):** 956 instances.
- **False Positives (NEG → POS):** 957 instances.

Recall is slightly higher for the negative class (~ 0.909) than for the positive class (~ 0.904).

This indicates the model is marginally more conservative when predicting positive sentiment, a common outcome when the linear boundary on averaged embeddings slightly favors the negative classification side.

Deep Learning Models: Results & Comparison

Table 10.26: Deep Learning Model Performance Comparison (Hotel Reviews)

Model	Accuracy	F1-score
BiLSTM	0.9373	0.9378
CNN	0.9261	0.9270

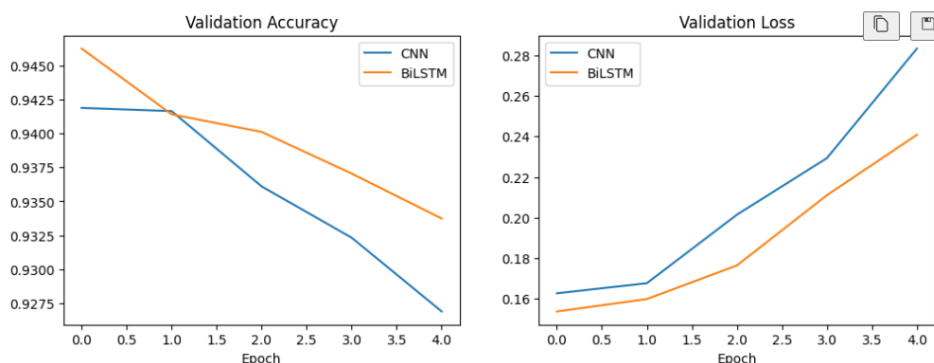


Figure 10.15: Visualization of CNN vs. BiLSTM performance on hotel reviews.

Table 10.27: Classification Report for CNN Model

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.9372	0.9134	0.9251	10,564
1 (Positive)	0.9155	0.9387	0.9270	10,563
Accuracy		0.9261		21,127
Macro Avg	0.9263	0.9261	0.9261	21,127
Weighted Avg	0.9263	0.9261	0.9261	21,127

Table 10.28: Classification Report for BiLSTM Model

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.9440	0.9299	0.9369	10,564
1 (Positive)	0.9309	0.9448	0.9378	10,563
Accuracy		0.9373		21,127
Macro Avg	0.9374	0.9373	0.9373	21,127
Weighted Avg	0.9374	0.9373	0.9373	21,127

The BiLSTM outperforms the CNN in both accuracy (~ 0.937 vs ~ 0.926) and F_1 -scores, demonstrating a superior ability to minimize misclassifications. While the CNN relies on local n -gram patterns through max-pooling—often leading to increased false positives from “positive-ish” phrases—the BiLSTM captures sequential order and global context. This is particularly effective for hotel reviews, where sentiment typically depends on the complex combination of clauses rather than isolated keywords.

Despite the BiLSTM showing a strictly better confusion matrix, validation curves for both architectures reveal significant gaps between training and validation performance. This suggests that while recurrent aggregation is more robust for this task, the models remain

sensitive to the current regularization and epoch settings, indicating further potential for tuning to mitigate overfitting.

Overall comparison

Table 10.29: Final Performance Summary: Deep Learning vs. Classical Baselines (Hotel Reviews)

Model	Accuracy	F1-score
BiLSTM	0.9373	0.9378
CNN	0.9261	0.9270
LogReg	0.9066	0.9063
LinearSVC	0.9064	0.9060

Evaluation of the Arabic reviews dataset demonstrates a shift in model hierarchy compared to previous experiments. While classical LogReg on L_2 -normalized AraVec FastText embeddings achieves a solid baseline of 0.906 accuracy and F_1 , it is surpassed by sequence-based neural architectures.

The neural baselines, utilizing task-trained embeddings, provide superior performance:

- **BiLSTM:** The top performer with ≈ 0.937 accuracy and 0.938 F_1 .
- **CNN:** Follows closely with ≈ 0.926 accuracy and 0.927 F_1 .

10.7 AraBERT

10.7.1 Tweets Dataset

Table 10.30: Performance Comparison of Classical Heads on AraBERT Embeddings (Tweets)

Model	Accuracy	F1-score
LinearSVC	0.8217	0.8250
LogReg	0.8121	0.8139

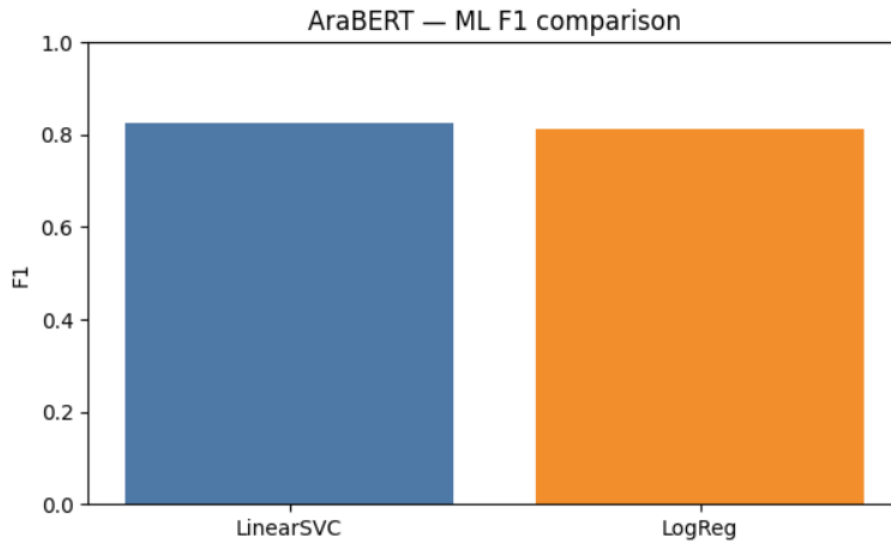


Figure 10.16: Performance metrics for LinearSVC vs. Logistic Regression using AraBERT features on tweet data.

Table 10.31: Classification Report for Best Model (AraBERT): LinearSVC

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.8400	0.7975	0.8182	158
1 (Positive)	0.8049	0.8462	0.8250	156
Accuracy		0.8217		314
Macro Avg	0.8224	0.8218	0.8216	314
Weighted Avg	0.8226	0.8217	0.8216	314

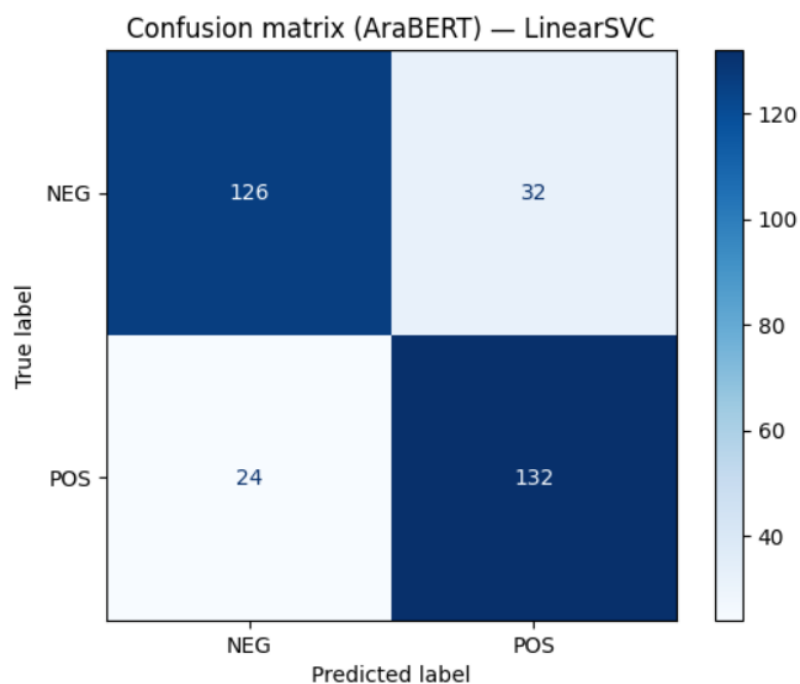


Figure 10.17: Confusion Matrix for LinearSVC with AraBERT (Tweet Dataset)

For this pipeline, tweets are encoded using **AraBERT** (v02), utilizing a masked mean pool of the last hidden layer and L_2 -normalization.

Two linear heads were evaluated on identical splits, revealing a distinct performance gap:

- **LinearSVC (Best):** Achieves approximately 82.2% accuracy and 0.825 positive-class F_1 .
- **Logistic Regression:** Trails by 1–1.5 percentage points with $\approx 81.2\%$ accuracy and 0.81 F_1 .

Error Analysis: The confusion counts for the top model (TN ≈ 126 , FP ≈ 32 , FN ≈ 24 , TP ≈ 132) indicate relatively balanced mistakes. However, there is a slight prevalence of false positives (negative reviews mislabeled as positive).

Conclusion: Contextual AraBERT embeddings paired with a simple linear classifier provide a robust baseline, with the SVM margin objective demonstrating a clearer advantage over logistic regression when processing these normalized sentence vectors.

10.7.2 Reviews Dataset

Table 10.32: Performance Comparison of Classical Heads on AraBERT Embeddings

Model	Accuracy	F1-score
LinearSVC	0.9239	0.9246
LogReg	0.9161	0.9165

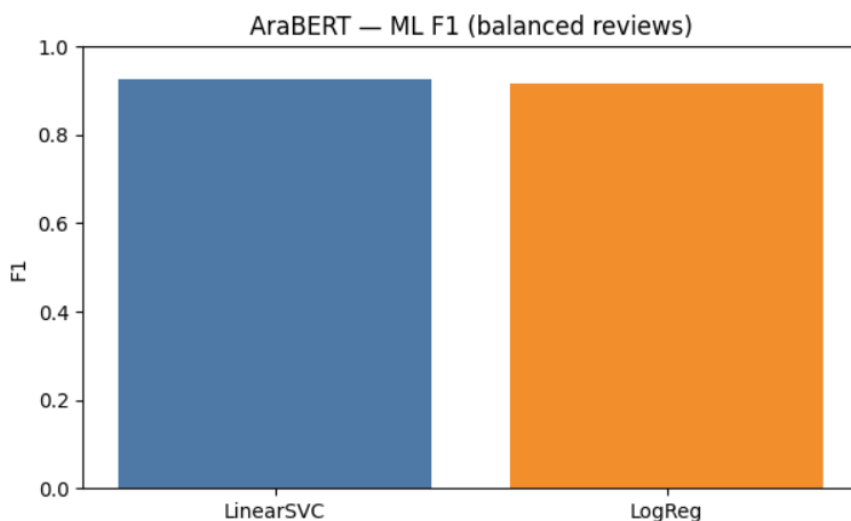


Figure 10.18: Performance metrics for LinearSVC vs. Logistic Regression using AraBERT features.

Table 10.33: Classification Report for Best Model (AraBERT): LinearSVC

Class	Precision	Recall	F1-Score	Support
0 (Negative)	0.9319	0.9147	0.9232	10,564
1 (Positive)	0.9162	0.9332	0.9246	10,563
Accuracy	0.9239			21,127
Macro Avg	0.9241	0.9239	0.9239	21,127
Weighted Avg	0.9241	0.9239	0.9239	21,127

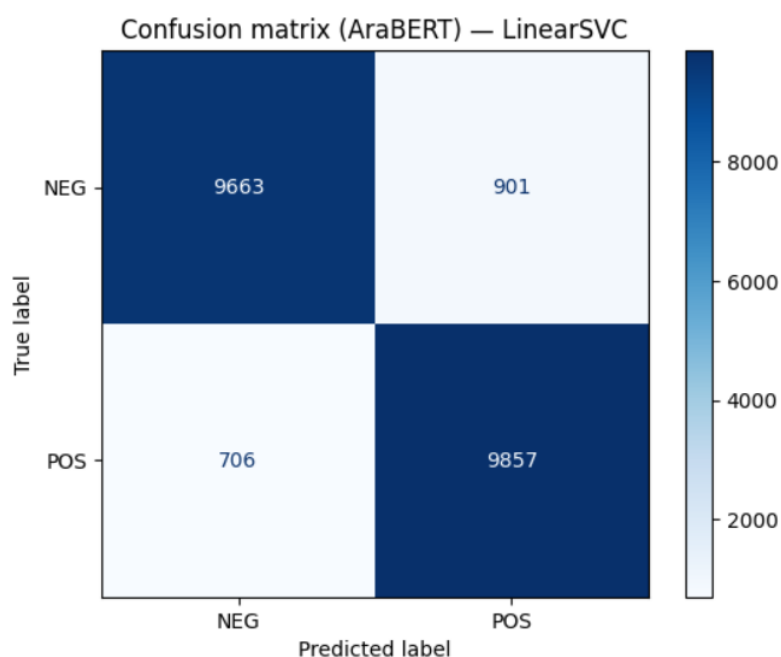


Figure 10.19: Confusion Matrix for LinearSVC with AraBERT Mean-Pooled Embeddings

10.8 Experimental Performance Synthesis

10.8.1 Binary Hotel Reviews

Evaluation across the hotel review pipeline reveals a competitive hierarchy where traditional sparse representations and sequence-modeling architectures yield the strongest overall results:

- **TF-IDF + Logistic Regression:** Achieves the highest overall performance with $\approx 94.2\%$ accuracy and a $0.942 F_1$ -score (`article_repro_pipeline_balanced_reviews.ipynb`), serving as the top classical baseline.
- **BiLSTM (Learned Embeddings):** Demonstrates robust sequence modeling with $\approx 93.8\%$ accuracy and a $0.938 F_1$ -score, strictly outperforming its 1D convolutional counterpart (CNN $\approx 92.7\%$ accuracy).
- **AraBERT (Mean-Pooled) + LinearSVC:** Provides strong contextual classification, yielding $\approx 92.4\%$ accuracy and a $0.925 F_1$ -score.
- **FastText (AraVec) + Linear Heads:** Underperforms relative to alternative pipelines, stabilizing in the $\approx 90.6\%$ performance regime, with LogReg marginally outperforming LinearSVC.

10.8.2 Binary Tweets

On the smaller, more volatile microblog split, the performance dynamics shift dramatically in favor of pre-trained contextual encodings over from-scratch representation learning:

- **AraBERT + LinearSVC:** Establishes the superior approach for this split, reaching $\approx 82.2\%$ accuracy and a $0.825 F_1$ -score (`article_repro_pipeline_embeddings.ipynb`).
- **FastText (AraVec) + LinearSVC:** Displays a notable performance drop, achieving $\approx 77.7\%$ accuracy and a $0.77 F_1$ -score.
- **Task-Specific Neural Baselines:** Both the `CNN` and `BiLSTM` architectures fall significantly behind the embedding-driven pipelines—landing within a lower ~ 0.67 – $0.69 F_1$ range—primarily due to the constraints of training standalone sequence layers from scratch under a limited data budget.

Chapter 11

Challenges

Arabic sentiment analysis remains difficult because the text is linguistically rich, domain-dependent, and often noisy. The main challenges observed in this project are:

- **Dialect and code-switching:** Tweets contain dialectal Arabic, named entities, hash-tags, and occasional foreign words, which makes vocabulary coverage harder for sparse models.
- **Informal spelling:** Repeated letters, missing diacritics, and inconsistent orthography increase the number of surface forms for the same word.
- **Ambiguity and negation:** Sentiment can depend on context, irony, or negation, which simple word-count features cannot always capture.
- **Domain differences:** Tweets are short and noisy, while hotel reviews are longer and often mix positive and negative statements in the same text.
- **Preprocessing trade-offs:** Strong normalization reduces sparsity, but it can also remove useful stylistic or dialectal cues.

These challenges explain why no single representation is ideal for every setting. Sparse models are efficient and competitive on longer review texts, FastText/AraVec helps with short noisy tweets, and AraBERT provides the strongest contextual understanding when enough computational resources are available.

Chapter 12

Conclusion

This project compared several approaches for Arabic sentiment classification across two different corpora: tweets from ASTD and hotel reviews from a balanced review dataset. The study covered preprocessing, feature extraction, feature selection, classical machine learning models, shallow deep learning models, FastText/AraVec embeddings, and AraBERT-based representations.

The full parameter-comparison experiment shows that fine-tuned AraBERT gives the best overall macro F_1 in both protocols, reaching 0.815 on ASTD and 0.800 on the four-class hotel-review task. This confirms the advantage of contextual Arabic pre-training for capturing morphology, negation, and semantic variation.

The results also show that representation choice depends strongly on the dataset. AraVec/FastText is highly effective on short, noisy tweets, where subword information helps recover sentiment cues from spelling variation and rare forms. In contrast, sparse BoW and TF-IDF features remain strong on longer hotel reviews because they preserve discriminative phrases and rating-specific expressions.

Classical models such as LinearSVC and Logistic Regression remain important baselines because they are fast, interpretable, and competitive. CNN and BiLSTM models can perform well on larger review data, but they are more sensitive to training size, regularization, and hyperparameters. Overall, the experiments show that Arabic sentiment analysis benefits from matching the representation and model family to the linguistic properties of each corpus.

Chapter 13

Future Work

Future improvements can extend the current work in several directions:

- tune AraBERT and neural models more extensively using additional epochs, learning-rate schedules, and stronger regularization;
- test dialect-aware Arabic models and compare them with AraBERT on informal tweet text;
- evaluate larger feature-selection grids and include confidence intervals across multiple random seeds;
- improve error analysis by manually inspecting false positives and false negatives for each dataset;
- explore ensemble methods that combine sparse lexical features with contextual embeddings.

Bibliography

- [1] ASTD Arabic sentiment tweets dataset and project data files.
- [2] Arabic hotel reviews dataset used in the project experiments.
- [3] AraBERT pretrained Arabic language model documentation and related publications.
- [4] Scikit-learn documentation for text vectorization, feature selection, and linear classifiers.
- [5] TensorFlow/Keras documentation for CNN, LSTM, and BiLSTM neural models.